20｜云时代的下一站：SaaS化与魔球建模法

2021-08-17 徐昊

《如何落地业务建模》 课程介绍 >

讲述：徐昊
时长 13:23 大小 12.27M

你好，我是徐昊。今天我们来聊聊 SaaS（Software as a Service）化。

按理说，SaaS 化与业务建模无关，本不应该出现在这门课程中。但是，SaaS 化却与云平台、微服务有着千丝万缕的联系。而且，我是通过建立一个模型解决了 SaaS 化服务设计的问题，所以勉强算是沾边吧。

那么今天就来讲一讲为什么我们需要在现在关注 SaaS 化，以及如何通过服务设计，对已经存在的产品进行 SaaS 化。

微服务真的能帮助创新吗？

很多企业上微服务，都是听信了这样的愿景：**微服务可以促进企业内创新，未来只需要编排存在的服务，就能满足一堆创新的需求。**微服务的确可以实现这个愿景，但是有个实现的前提：**微服务需要以开放服务的形式，被其他应用或服务调用**。

所谓开放服务，即：这个服务不是为某个特定应用或服务构建的，而是面向生态中所有可能的客户端构建的。

微服务并不是服务化的应用后台，它封装了企业的业务能力。任何需要用到这个业务能力的应用，都会对这个服务进行调用。因而微服务与应用后台在概念上就存在巨大的差异。应用后台仅仅服务于应用前台，只要满足前台的需求就可以了。而微服务不仅需要服务于当前已知的应用，还要服务于未来可能需要用到对应业务能力的应用。

除此之外，我需要特别强调，**开放还指的是，并不是专门为生产环境中的应用或服务构建的，而是面向所有环境内的应用或服务构建的。**也就是说，这个微服务是否能够允许，围绕它展开试验和创新，同时还不影响生产环境下的正常使用。

说句题外话，我见过很多过度拟合的微服务（Overfitting Microservices）。过度拟合这个概念是我从机器学习那里借来的，表示那些没有封装业务能力，过度考虑某些应用场景的服务。这也算是伪微服务的一种形态。

再说一句题外话，我同事 Zhamak Dehghani（Data Mesh 概念的初创者）曾经在我们某次酒吧扯淡中，描述过一种微服务的机缘凑巧（Serendipity in Microservices）。意思是在微服务完全成为开放式服务生态之后，你无法预知会有什么样的创新出现。因而在机缘巧合之下，会有很多意外之喜。我完全同意这样美好的愿望，而我唯一的顾虑在于，serendipity 本义一定指的是好事，但在开放式服务生态中，却没法保证发生的一定是好事。

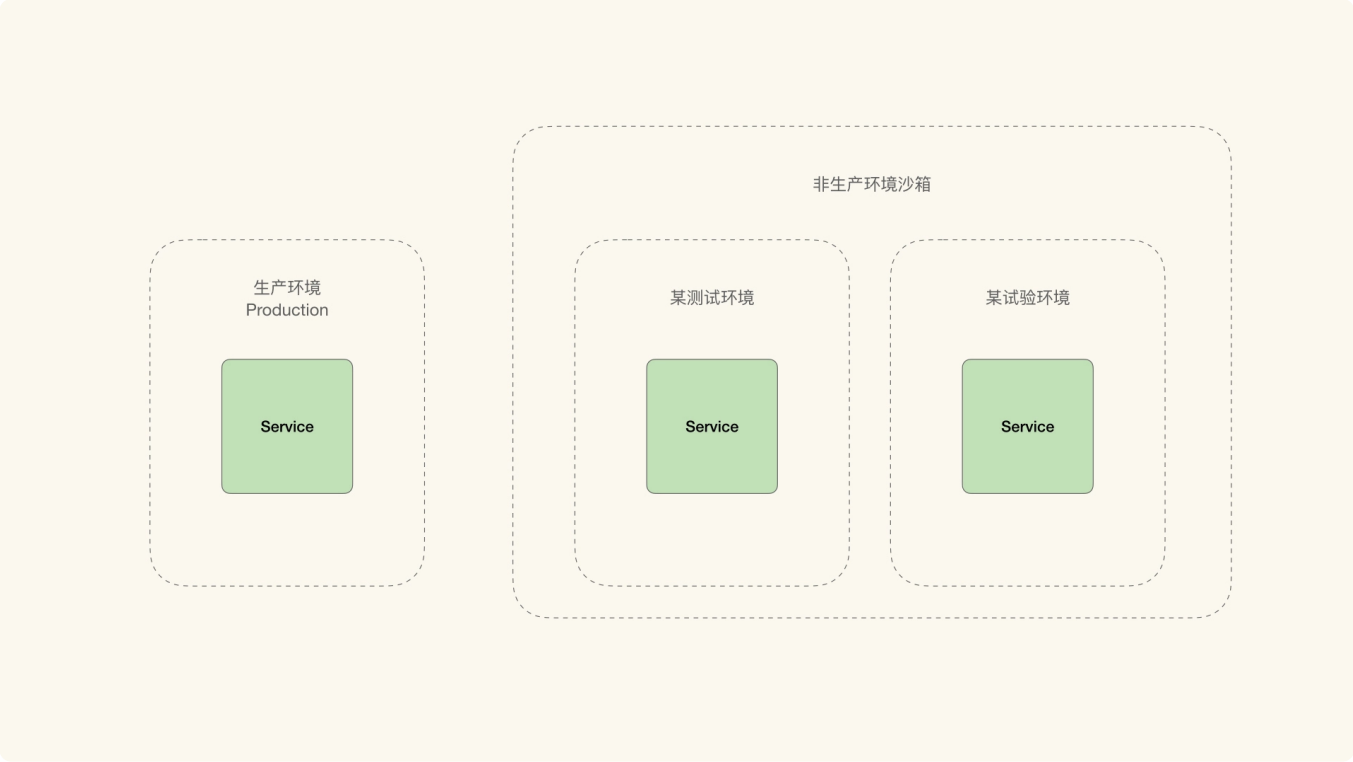
言归正传。总之，我们需要将微服务构造成开放式服务，那么有两种策略可选：完全开放式服务和 SaaS 化的开放服务。

我们先来看完全开放式服务。也就是对于某个版本的服务，在企业内生态中存在一个唯一的、开放的、可供任何人访问的实例。这时候，这个服务就像是微博、搜索引擎等互联网开放服务一样。只要具有对应的访问权限（通常是注册用户），那么就可以毫无顾忌地对这个服务进行访问。

对于只读性服务，这种策略问题不大。比如提供企业内所有员工公开信息的服务，再比如提供所有产品信息的产品目录服务等。在这种情况下，只需要保证足够的弹性，以应对试验与创新带来的流量即可。

而对于会留下业务痕迹的业务系统，这种完全开放式服务的策略可能就不是那么适用了。来自于试验和创新的流量会在生产系统中留下痕迹，也可能会影响生产环境的正常使用。那么这时候，采用 SaaS 化的开放服务可能就是更好的选择。

这时候被 SaaS 化的，就是这个服务的沙箱环境（Sandbox Environment）。也就是说，在任何非生产环境的请求，都会在沙箱环境中获得独立的、与生产环境隔离的专有环境。如图所示：



如果存在需要借助这个服务进行某种实验或创新的场合，那么都可以通过 SaaS 化的沙箱获得自己专用的服务实例，完成所需的实验。同样，如果存在另一个服务需要与这个服务交互，那么上线前最终的验证与测试，也可以在沙箱环境中完成。

这里你可能会问，对于测试的情况，难道不能通过对契约打桩（Contract Stub）来实现吗？当然可以。只是通过 SaaS 化的沙箱，可以提供更接近生产环境的环境。

需要注意的是，有一种特殊的 SaaS 化形式，我称之为冷 SaaS。也就是服务团队不提供运行沙箱的基础设施硬件，而仅仅提供相应的镜像和脚本。这时候就需要沙箱环境的团队，

可以在自己的基础设施上构造沙箱环境。因为云时代的基础设计与镜像等价（Infrastructure as Code，IaC 的一种形式），那么我们仍然可以在概念上，将这种方式看作是 SaaS。

因而，当我们完成了微服务改造之后，想要发挥微服务承诺的美好未来，总是要通过 SaaS 化将服务开放化的。

服务还是选项？

云计算的普及也是推动我们需要关注 SaaS 的一个重要因素。请考虑这样一个例子，针对某个应用或服务，我们希望可以对其进行云化处理，也就是使其具有水平扩展的能力。那么在假设这个应用或服务已经符合云化架构的诸多特点之后，最简单的做法是，将其放置于一个弹性边界之内，并通过弹性负载均衡处理相应的流量。当流量增加时，我们就在弹性边界内增加实例，以应对多出来的流量。

正如我们前面课程讲到的，这是一个典型的云化策略。但是这种策略有个问题，它会**假设所有的用户都有一样的服务质量需求**。这就相当于什么呢？这就相当于，当你买咖啡的时候，只有超大杯；当你发快递的时候，只有昂贵的次日达；当你选手机卡套餐的时候，只有 699 包月这一档。

当然你会问，超量满足客户的需求不好吗？如果你真能超量收钱的话，就当我没说。但绝大多数情况下，是为了服务好每一个用户而付出了超量的成本。所以这种云化策略就是一种昂贵的云化策略。

以可用性为例，我们都知道从 99.9999% 提升到 99.99999% 要付出巨大的成本代价。反之，从 99.99999% 降低到 99.99% 就能获得巨大的成本缩减。那么所有用户都需要一样的可用性吗？

可以再考虑一下前面微服务的例子，生产环境和沙箱环境对于可用性的要求明显不同。而且就算在生产环境中，不同消费者对于可用性的诉求其实也不尽相同。那么，作为功利主义架构师（Utilitarianism Architect），我自然要问，**既然实际上消费者有不同的可用性诉求，那么什么样的云化策略，可以在实质上不影响服务质量的前提下，降低成本呢**？

答案是 SaaS 化，但是**并不是围绕着服务或应用，而是围绕着选项（Offering）**。

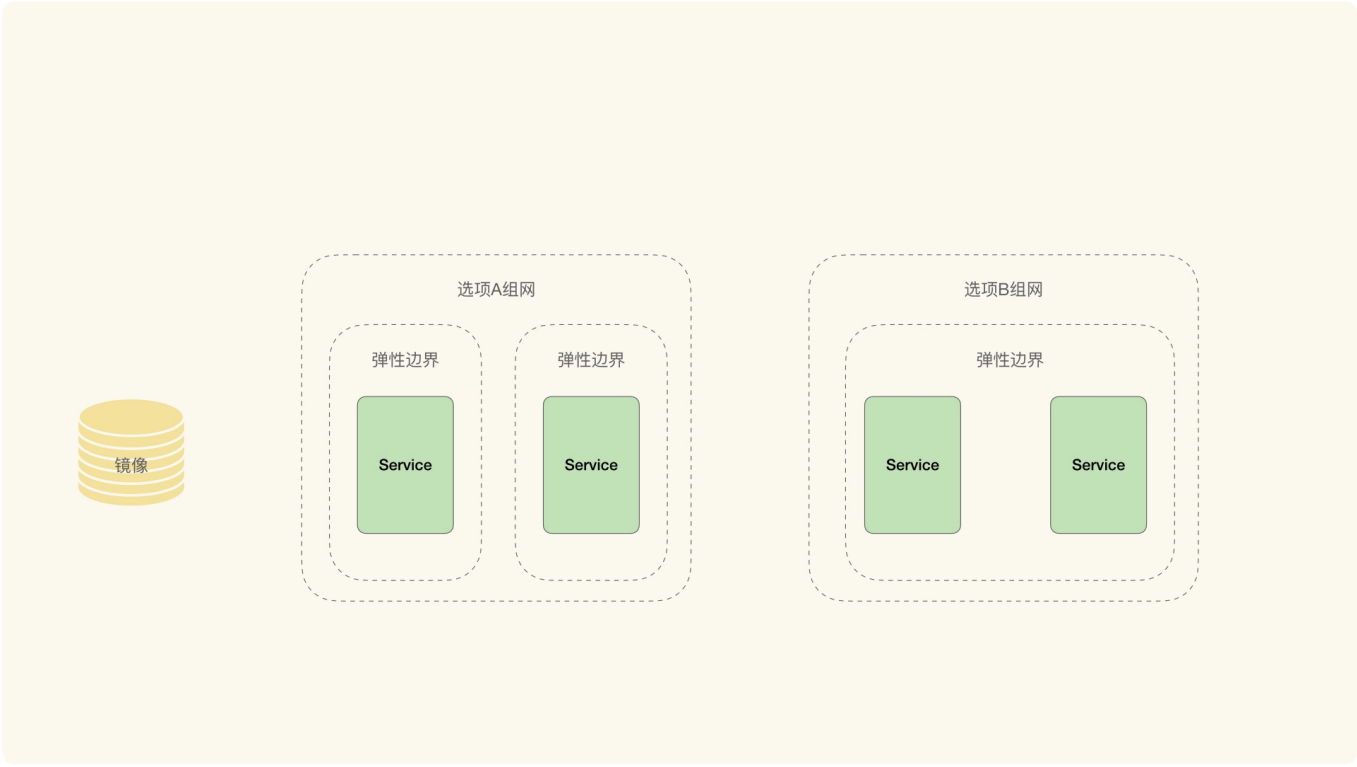
之所以产生了昂贵的云化策略，原因就在于我们忽略了不同用户对于服务质量（可用性只是其中之一）的差异化需求，而仅仅考虑了服务或应用能力上的差异。那么，我们就需要将服务质量重新纳入考量。于是我们可以这么定义：

选项（Offering）= 能力（Capabilities）+ 服务质量（Service-Level Agreement/Non Functonal Requirement，SLA/NFR）

我们随便举个例子，比如快递，能力都是一样的，把包裹从 A 地运到 B 地。那么根据不同的服务质量，我们就可以定义出不同的选项：同城当天可达的是当日达；第二天保证送到的是次日达；三到五天的是标快；七到十天的是特惠；等等。

从运营的角度上讲，不同的选项实际代表了不同的运营成本。比如特惠可能会使用便宜的运力，像大卡车；而次日达则可能会使用昂贵的运力，像全货机。那么对应到云环境下，不同的服务质量实际上表示了不同的组网环境。

就以可用性为例，当我们选择不同的云平台选项时，基础可用性本身就存在差异，从多个 9 到不到 90% 都有可能。那么我们可以在不同的组网上，实现我们所需的可用性需求。然后当有流量进入时，我们可以按照不同流量的种类（比如开通的服务），在不同的组网上启动镜像即可。如下图所示：

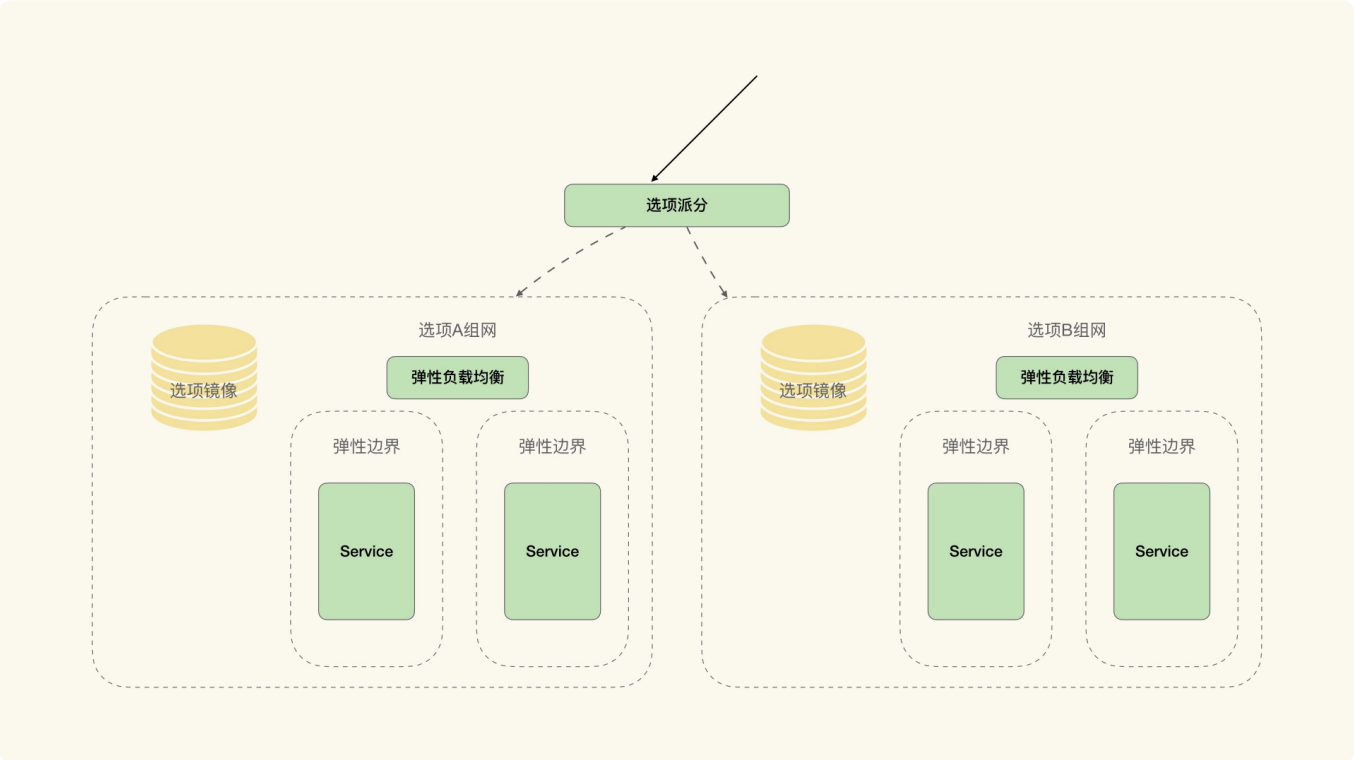


在图中的例子里，对于选项 A，我们可能要求 5 个 9 的可用性，那么我们在 6 个 9 的云平台选项基础上，采用专用服务集群策略（dedicated services）实现 5 个 9 的可用性；而对于选项 B，可能只需要 1 个 9，那么我们可以在 2 个 9 的云平台选项基础上，采用共享服务策略（shared services）实现 1 个 9 的需求。

但无论是 5 个 9 还是 1 个 9，它们都是根据相同的镜像来获得服务实例的。因而在不同的组网之上，选项 A 和选项 B 的能力仍然是相同的，差异只存在于服务质量。

我们就可以根据消费者的不同诉求分派流量。那么选择选项 A 的消费者，可能是生产环境中的关键消费者，对于我们的服务存在关键路径依赖；而选择选项 B 的消费者，可能只是在边角功能中用到了我们的服务，而且并不在意我们的服务质量。或是实验性项目，或是测试沙箱等等。通过分流选项 B 的流量，我们节约了大量的运营成本。我们相当于围绕着不同的选项构建了 SaaS 化的云化策略。

那么你可能会问，在不同的选项中，能力可以不同吗？答案是可以的（当然这对于软件交付的策略、变化点的识别、架构的策略都提出了更高的要求。但由于不是我们这门课的主题，这里就不展开了）。于是我们可以得到一个通用的 SaaS 化的部署模式：



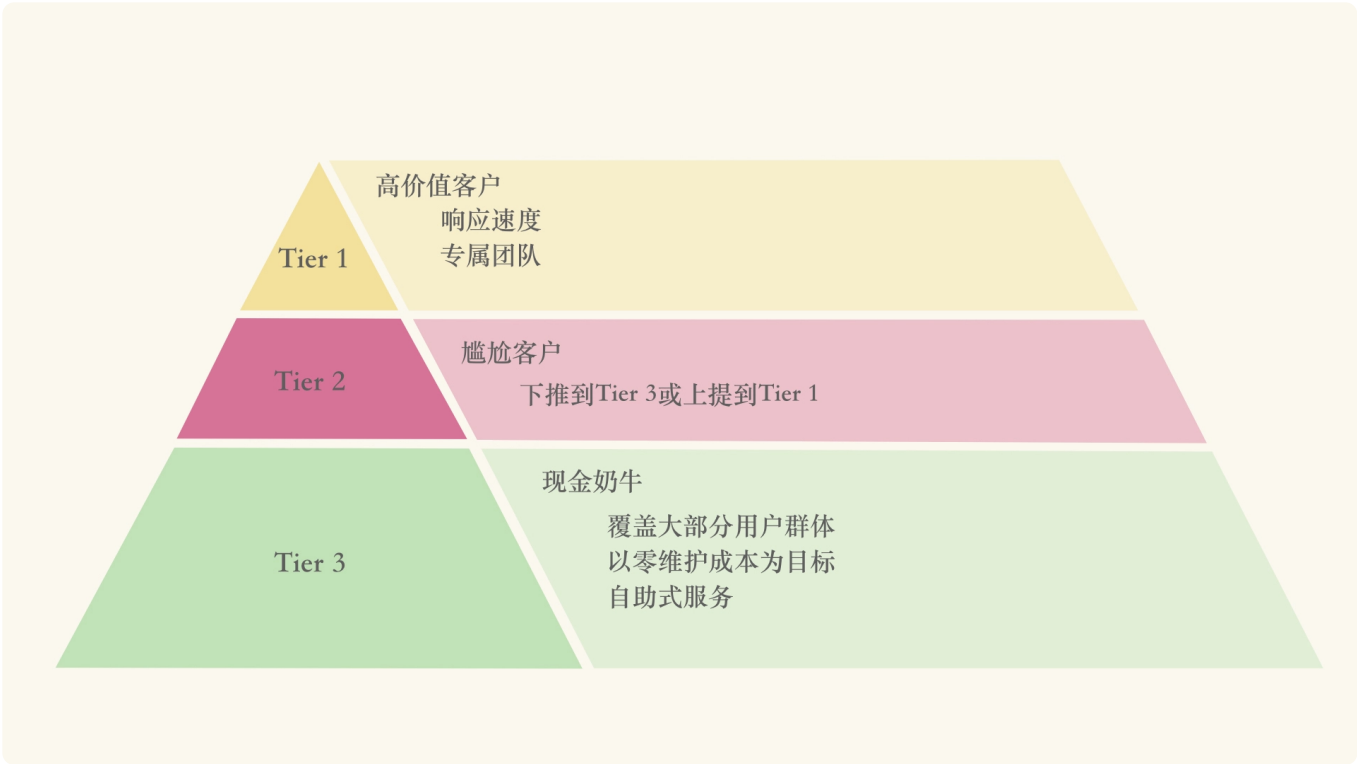
你自然可以脑补出在 K8S、Rancher 上的各种具体实现，我就不多说了。我的很多客户总是对实现 SaaS 化的技术细节很感兴趣，所以我每次都免费送他们这个图。因为这个图完

全不值钱。毕竟决定 SaaS 化成败的关键，在于如何正确地设计选项，否则我们又怎么会知道在不同的组网中该放什么镜像，要去实现哪些服务质量呢？

需要几个选项？

那么我们要怎么设计选项呢？又需要几个选项呢？答案很简单，对于任意 SaaS 化的应用或者服务，都需要三类选项以应对三类客户：

1. 高价值客户：这类客户除了看重服务质量外，更看重响应速度。类比到市面上可以见到的 SaaS 服务，也就是 GitHub 企业版、Salesforce 企业版这样的。对于这类客户，快速满足他们的个性化诉求，才是赢得青睐的关键。因而，通常 SaaS 产品，不光会采用单独独立的组网环境，还会提供专门的服务团队，以满足用户对于响应速度的要求。
2. 现金奶牛：这类客户应该覆盖大部分的用户群体，比如 GitHub 免费用户，或是 Salesforce 标准版这样的用户。对于这类客户，目标就是降低成运营成本。比如通过自动化脚本实现低成本运维。再比如，通过自助式服务来解决服务开通等问题。
3. 尴尬客户：这类客户价值不高，成本又减不下来。食之无味弃之可惜，庙小妖风大池浅王八多，都是这类客户的真实写照。解决办法通常是将其提升为高价值客户，或者简化为现金奶牛。但也需要注意的是，这些客户是 SaaS 化应用持续改进的重要源头。



这就是我发明的**魔球 SaaS 选项建模法**（Moneyball SaaS Offering Modeling）。从名字你也能看出来，灵感来自篮球中的魔球理论：三分价值高，上篮把握性大，中距离很尴尬。所以，要尽可能地将出手转化为上篮或者三分，避免中距离。同样的方法，可以帮助我们建模 SaaS 选项：**将客户尽可能地转化为现金奶牛或者高价值客户，避免尴尬客户，从而得到一个简单直接的选项设计。**

划分了客户群体类别之后，就可以进行选项设计了。这个时候，就需要考虑客户在能力和服务质量上的差异化诉求了。你可以在具体场景中自由发挥。

最后再顺便说一点。魔球建模中的三类客户，也代表了一种将存量系统 SaaS 化的变革策略。我们可以将现有系统和现有的服务质量，建模成遗留选项（Legacy Offering），然后将它放置在尴尬客户的 Tier 2。那么现在我们 100% 的用户都是尴尬客户了。

这个策略虽然不太妙，但是说明我们潜力无穷啊！于是我们可以建立相应的变革策略，从中分化出高价值客户和现金奶牛，并用新的选项去覆盖他们，这里可以采用产品设计和产品演进的思路推进变革。

鉴于产品设计和产品演进也不是我们这门课的核心内容，这里就不具体展开了。需要了解这方面信息的，可以找我司 CX（Customer Experiences）的产品专家们。

小结

这节课我们从微服务和云化策略入手，展望了它们后续的发展。发现殊途同归，最后都需要 SaaS 化能力。所以我说，无论是微服务还是云时代，下一站都会是 SaaS 化。当 SaaS 化成为我们默认的运营和架构风格，我们就进入了云时代的下一章。

在下一章中，不仅仅是业务建模，产品化设计、产品化运维、服务消费体验、生态体验都将会进入我们的视野。到那时，技术的景致（landscape）将会大有不同。我希望你能以一个开放的心态，迎接下一章。特别是，希望你能从现在开始，补充产品与体验设计相关的知识和技能。

让我们到那时再相见！See you on the other side！

思考题

最后一节课了，来畅想一下吧。如果进入下一章，你会怎样？

到这里，我们专栏的正文内容就全部结束了，感谢你一直以来的陪伴！还有一篇“结束语”马上会和你见面，我们下一讲再见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 19 | 如何将模型实现为微服务？

下一篇 结课测试题 | 这些业务建模知识你都掌握了吗？

精选留言 (6)

写留言

鸭补一生如梦
2021-08-18

产品是企业价值的载体，是企业与消费者进行价值交换，实现企业增长赋能的重要容器。

产品架构与技术架构是企业价值的一体两面，不知徐老师能否推荐产品架构的专栏，让我们继续进入“下一章”。谢谢

展开

Williamleelol

2021-08-18

催更徐老师八叉说～(￣▽￣～)~

展开

Jxin

2021-08-17

1.都说波士顿矩阵过于简单粗暴，对其的优化一般也都是升维，比如通用电气矩阵。老师这个魔球直接再给降维了一波。
2.尴尬客户这一块，上无门下无用，离队可能也会是个选项。毕竟向下能满足不会处于尴尬，商业模式运营向好也不会停留在尴尬。就是因为收益不景气加差异化问题多才处在尴尬阶段，而这个现状上下无门。...

展开

作者回复: 一般也就是维持 然后结束生命周期

Geek_de900a

2021-08-17

大致学习了一遍，感觉没完全整明白，，老师，要不要考虑开个训练营。

编辑回复: “大致”可不行哈，得“仔细”。
“一遍”也不行哈，得“三遍”起。

Janson

2021-08-17

抛开技术看，在SaaS化策略执行时还是主要从企业运营考虑（利润和成本）

云师兄

2021-08-17

魔球 SaaS 选项建模法，有意思

展开